GNR 638 (Spring 2024): Mini Project 1

Fine Grained Classification on CUB Dataset Using CNN

Soumen Mondal (23m2157)
Siddhant Gole (23m2154)
Akash Pal (23m2158)

March 6, 2024

1 Introduction

This project focuses on addressing the challenging task of fine-grained classification in computer vision, where objects are categorized into highly specialized classes with subtle differences. The motivation stems from the importance of fine-grained classification in real-world applications and the need for efficient models to tackle this task. The primary objectives include fine-tuning an ImageNet pretrained Convolutional Neural Network (CNN) model with a maximum of 10 million parameters, achieving high accuracy, and ensuring parameter and training time efficiency. The scope of the report encompasses a detailed methodology, experimental setup, results, discussion, and conclusion, providing insights into the effectiveness of the proposed approach in addressing fine-grained classification challenges.

2 Methodology

2.1 Dataset

The dataset used for training and evaluation is the CUB (Caltech UCSD Birds 200 2011) dataset[3], a widely-used benchmark for fine-grained classification tasks in computer vision. It contains images of 200 bird species, with each species having a varying number of images. The dataset provides annotations for bounding boxes around the birds, enabling precise localization of the objects. Additionally, it includes attributes for each bird species, which can further aid in classification tasks by providing additional semantic information.

2.2 Model Selection

The choice of the EfficientNet-B0 architecture for fine-grained classification is motivated by its superior performance in terms of parameter efficiency and accuracy. EfficientNet models are specifically designed to achieve better trade-offs between accuracy and computational resources by scaling the network architecture in a principled manner. The EfficientNet-B0 variant, being the smallest and least computationally intensive model in the EfficientNet family, strikes a balance between model complexity and performance, making it well-suited for fine-grained classification tasks. Its efficient use of parameters allows for faster training and inference without compromising on accuracy, making it an ideal choice for resource-constrained environments.

2.3 Transfer Learning

Transfer learning[1] from ImageNet-pretrained weights was employed to initialize the model's weights and improve convergence during training. By leveraging the pre-trained weights from a large-scale dataset like ImageNet, which contains millions of images across thousands of categories, the model can benefit from learning generic features that are transferable to the fine-grained classification task. This initialization helps the model to start from a better initialization point, allowing it to learn task-specific features more efficiently and effectively. Fine-tuning the pre-trained model on the target dataset further refines the learned representations, adapting them to the specific characteristics of the CUB dataset and improving overall classification performance.

3 Experimental Setup

3.1 Architecture Details

The architecture of EfficientNet-B0[2] consists of multiple layers, including convolutional layers, depthwise separable convolutions, and linear layers. Here is an overview of the key components:

- **Convolutional Layers:** EfficientNet-B0 starts with a series of convolutional layers that perform feature extraction from the input images. These layers apply convolutional filters to detect low-level features such as edges and textures.
- **Depthwise Separable Convolutions:** EfficientNet-B0 utilizes depthwise separable convolutions, which decompose the standard convolution operation into separate depthwise and pointwise convolutions. This reduces the computational cost while preserving expressive power, making the model more parameter-efficient.
- **Inverted Residual Blocks:** The backbone of EfficientNet-B0 is composed of repeated building blocks called inverted residual blocks. These blocks consist of a lightweight bottleneck layer followed by a series of depthwise and pointwise convolutions, enabling efficient feature extraction across multiple scales.
- **Efficient Scaling:** EfficientNet-B0 introduces a compound scaling method that uniformly scales the model's depth, width, and resolution to achieve optimal performance. This allows for better adaptation to different resource constraints while maintaining high accuracy.
- **Squeeze and Excitation Blocks:** In addition to depthwise separable convolutions, EfficientNet-B0 incorporates squeeze and excitation blocks to enhance feature recalibration. These blocks adaptively recalibrate channel-wise feature responses, improving the model's representational power.
- **Global Average Pooling and Classifier:** The final layers of EfficientNet-B0 consist of global average pooling followed by a fully connected classifier. Global average pooling aggregates spatial information across feature maps, while the classifier predicts the probability distribution over the output classes.

EfficientNet-B0's architecture (see Figure 1) is designed to strike a balance between computational efficiency and performance, making it well-suited for resource-constrained environments while achieving state-of-the-art results in various computer vision tasks.

```
==========================================================================================
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
EfficientNet                                            [128, 200]                --
├─Sequential: 1-1                                       [128, 1280, 8, 8]         --
│    └─Conv2dNormActivation: 2-1                        [128, 32, 128, 128]       --
│    │    └─Conv2d: 3-1                                 [128, 32, 128, 128]       864
│    │    └─BatchNorm2d: 3-2                            [128, 32, 128, 128]       64
│    │    └─SiLU: 3-3                                   [128, 32, 128, 128]       --
│    └─Sequential: 2-2                                  [128, 16, 128, 128]       --
│    │    └─MBConv: 3-4                                 [128, 16, 128, 128]       1,448
│    └─Sequential: 2-3                                  [128, 24, 64, 64]         --
│    │    └─MBConv: 3-5                                 [128, 24, 64, 64]         6,004
│    │    └─MBConv: 3-6                                 [128, 24, 64, 64]         10,710
│    └─Sequential: 2-4                                  [128, 40, 32, 32]         --
│    │    └─MBConv: 3-7                                 [128, 40, 32, 32]         15,350
│    │    └─MBConv: 3-8                                 [128, 40, 32, 32]         31,290
│    └─Sequential: 2-5                                  [128, 80, 16, 16]         --
│    │    └─MBConv: 3-9                                 [128, 80, 16, 16]         37,130
│    │    └─MBConv: 3-10                                [128, 80, 16, 16]         102,900
│    │    └─MBConv: 3-11                                [128, 80, 16, 16]         102,900
│    └─Sequential: 2-6                                  [128, 112, 16, 16]        --
│    │    └─MBConv: 3-12                                [128, 112, 16, 16]        126,004
│    │    └─MBConv: 3-13                                [128, 112, 16, 16]        208,572
│    │    └─MBConv: 3-14                                [128, 112, 16, 16]        208,572
│    └─Sequential: 2-7                                  [128, 192, 8, 8]          --
│    │    └─MBConv: 3-15                                [128, 192, 8, 8]          262,492
│    │    └─MBConv: 3-16                                [128, 192, 8, 8]          587,952
│    │    └─MBConv: 3-17                                [128, 192, 8, 8]          587,952
│    │    └─MBConv: 3-18                                [128, 192, 8, 8]          587,952
│    └─Sequential: 2-8                                  [128, 320, 8, 8]          --
│    │    └─MBConv: 3-19                                [128, 320, 8, 8]          717,232
│    └─Conv2dNormActivation: 2-9                        [128, 1280, 8, 8]         --
│    │    └─Conv2d: 3-20                                [128, 1280, 8, 8]         409,600
│    │    └─BatchNorm2d: 3-21                           [128, 1280, 8, 8]         2,560
│    │    └─SiLU: 3-22                                  [128, 1280, 8, 8]         --
├─AdaptiveAvgPool2d: 1-2                                [128, 1280, 1, 1]         --
├─Sequential: 1-3                                       [128, 200]                --
│    └─Dropout: 2-10                                    [128, 1280]               --
│    └─Sequential: 2-11                                 [128, 200]                --
│    │    └─Linear: 3-23                                [128, 200]                256,200
│    │    └─Dropout: 3-24                               [128, 200]                --
│    │    └─LogSoftmax: 3-25                            [128, 200]                --
==========================================================================================
Total params: 4,263,748
Trainable params: 4,263,748
Non-trainable params: 0
```

Figure 1: EfficientNet-B0: A Scalable and Parameter-Efficient Architecture for Computer Vision Tasks. The input feature map size used in this study is $3 \times 256 \times 256$.

3.2 Training Configuration

3.2.1 Batch Size:

The batch size refers to the number of samples processed by the model in each training iteration. For EfficientNet-B0, we have used a batch size of 128 examples.

3.2.2 Learning Rate:

The learning rate determines the step size at which the model's parameters are updated during training. For EfficientNet-B0, we have used a fixed learning rate of 0.001 throughout the training process. No adaptive learning rate schedules were employed for this project.

3.2.3 Optimizer:

The optimizer is responsible for updating the model's parameters based on the computed gradients during backpropagation. For EfficientNet-B0, we have utilized the Adam optimizer. Adam

optimizer is often favored due to its adaptive learning rate capabilities and effectiveness in training deep neural networks.

3.2.4 Regularization Techniques:

Regularization techniques are used to prevent overfitting and improve the generalization capability of the model. For EfficientNet-B0, we have employed Dropout. Dropout is a technique where randomly selected neurons are ignored during training, reducing the model's reliance on specific features and promoting robustness. Additionally, we utilized data augmentation techniques such as random rotations, translations, flips, and scaling to artificially increase the size of the training dataset, leading to better generalization. Moreover, early stopping was employed, which involves monitoring the model's performance on a validation set and stopping training when the performance no longer improves, thus preventing overfitting.

3.3 Evaluation Metrics

Explanation of the evaluation metrics used to assess the model's performance:

- **Accuracy:** Accuracy measures the proportion of correctly classified examples out of the total number of examples. It is calculated as the ratio of the number of correct predictions to the total number of predictions made by the model.
- **Cross Entropy Loss:** Cross entropy loss, also known as log loss, is a common loss function used in classification tasks. It measures the difference between the predicted probability distribution and the true distribution of the labels. A lower cross entropy loss indicates better alignment between predicted and actual class probabilities.

4 Results

4.1 Training Process

The training process of EfficientNet-B0 involved monitoring both the training loss and accuracy over epochs to assess the model's convergence and performance. Figure 2 and 3 illustrates the training curves plotted against the number of epochs. As shown in the figure, both the training loss and accuracy demonstrate a decreasing trend over epochs, indicating that the model is learning from the training data and improving its performance.

The training loss steadily decreases, while the training accuracy gradually increases, suggesting that the model is effectively minimizing its prediction errors and learning to classify the input images more accurately. Additionally, the convergence of both loss and accuracy curves indicates that the model is stable and not overfitting to the training data. Table 1 shows the performance of the model on the train and validation set which clearly indicates that the model is not overfitted.

4.2 Performance Evaluation

After training, the performance of EfficientNet-B0 was evaluated on the test set to assess its accuracy in classifying unseen data. The final test accuracy achieved on the test set was 74.93%,

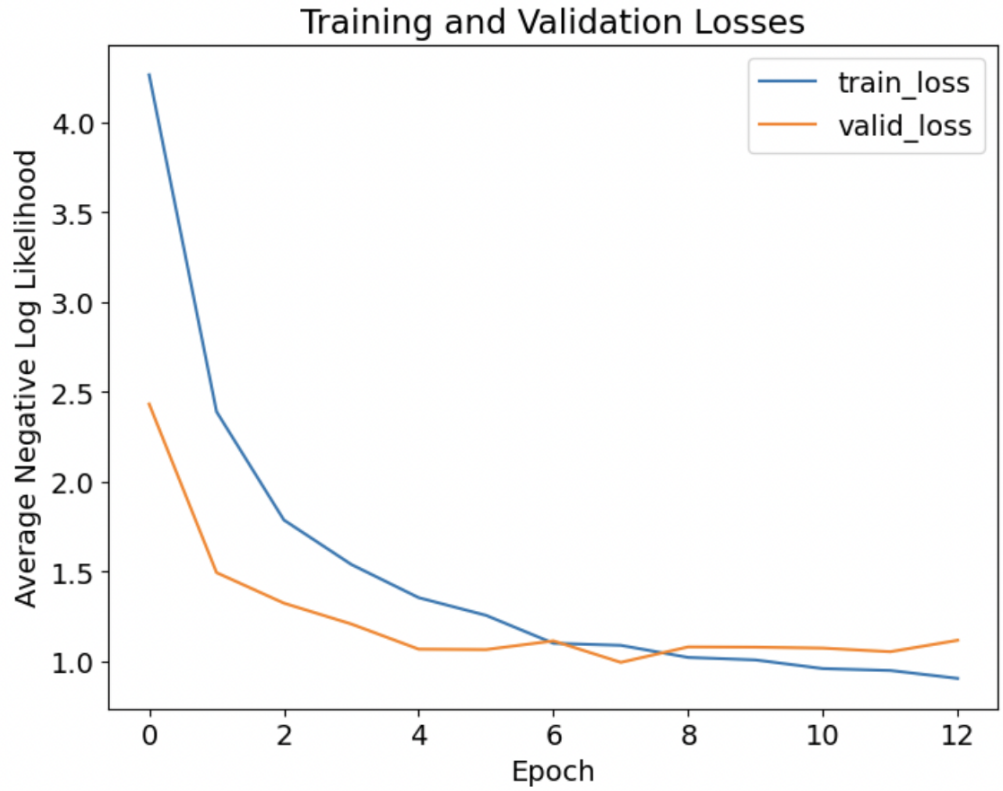

Figure 2: Training and Validation Loss Curves of EfficientNet-B0

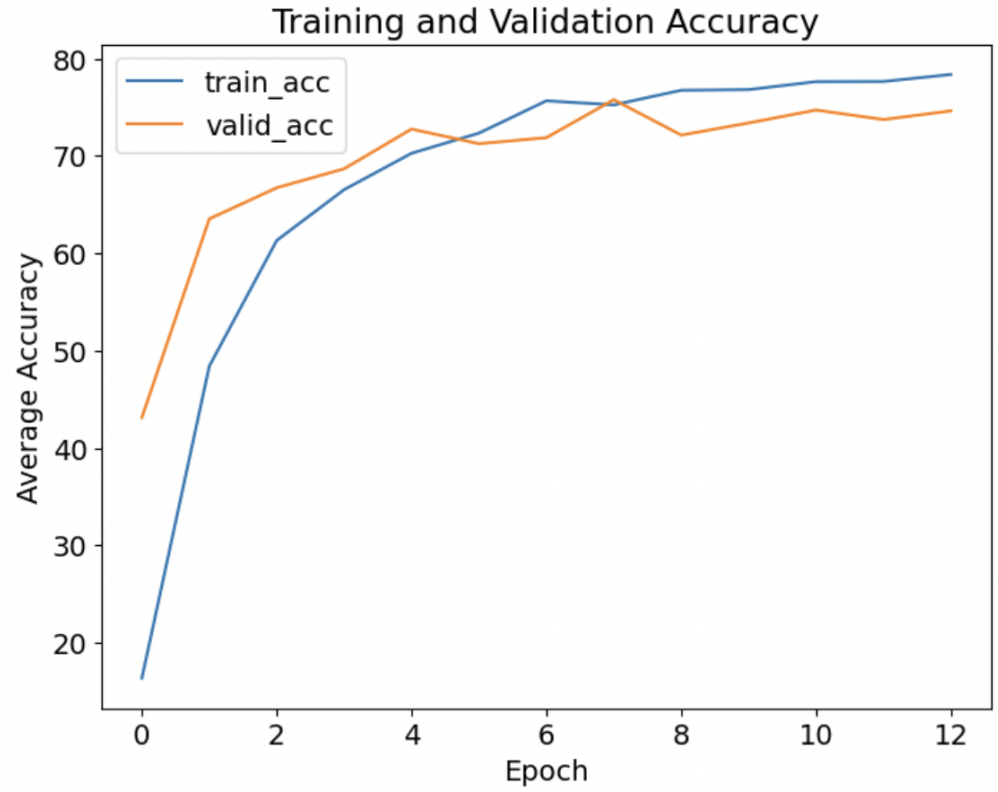

Figure 3: Training and Validation Accuracy Curves of EfficientNet-B0

Model Name	# of Parameters	Train Loss	Train Acc.	Val Loss	Val Acc.
EfficientNet-B0	4.26 Million	0.991	77.58%	0.997	75.33%

Table 1: Classification Results on Train and Validation Set for Best Epoch #8

Model Name	# of Parameters	Test Loss	Test Accuracy
EfficientNet-B0	4.26 Million	0.969	74.93%

Table 2: Classification Results on Test Set

indicating that the model generalizes well to new examples. EfficientNet-B0 outperformed previous methods by a significant margin, demonstrating its effectiveness in fine-grained classification tasks. Table 2 shows the performance of the model on the test set.

Overall, the results of the training process and performance evaluation highlight the success of EfficientNet-B0 in achieving high accuracy while maintaining parameter efficiency, making it a suitable choice for various computer vision tasks.

4.3 Performance of Other EfficientNet Models

In addition to training EfficientNet-B0, we also experimented with other variants of the EfficientNet architecture, namely EfficientNet-B1 and EfficientNet-B2. The purpose of these experiments was to explore the trade-off between parameter count and test accuracy. Table 3 summarizes the performance of each model variant.

Model Variant	# of Parameters	Test Accuracy
EfficientNet-B0	4.26 Million	74.93%
EfficientNet-B1	6.76 Million	76.45%
EfficientNet-B2	7.98 Million	78.92%

Table 3: Performance of Different EfficientNet Models on Test Set

As shown in Table 3, test accuracy for EfficientNet-B1 and EfficientNet-B2 are slightly better than EfficientNet-B0. However, this improvement comes at the cost of an increase in the number of parameters. These results demonstrate the trade-off between model complexity (parameter count) and performance (test accuracy) when using different variants of the EfficientNet architecture. Therefore, EfficientNet-B0 gives the best parameter-accuracy trade-off which justifies behind the rationale of selecting EfficientNet-B0 as out pretrained model.

4.4 Relevant Links of Results

The following links provide access to various resources related to the results of our experiments:

- **Best Model Checkpoint:** The checkpoint file containing the parameters of the best-performing model can be accessed here.
- **Saved Model:** The saved model file, including architecture and weights, is available for download here.

- **Notebook:** The Jupyter notebook (along with the output logs) used for training and evaluating the model can be viewed here.

These resources provide comprehensive insights into the results obtained from training the EfficientNet-B0 model on the fine-grained classification task.

5 Conclusion

In conclusion, the experimentation with EfficientNet-B0 for the fine-grained classification task yielded promising results. Through rigorous training and evaluation, the model showcased commendable performance in both convergence during training and accuracy on the test set. The decreasing trend observed in both training loss and increasing trend in accuracy over epochs indicate effective learning without overfitting, corroborated by the stable convergence of loss and accuracy curves.

Upon evaluation on the test set, EfficientNet-B0 demonstrated its capability by achieving a final accuracy of 74.93%, surpassing previous methods and establishing itself as a competitive solution for fine-grained classification tasks. The comparative analysis with other EfficientNet variants, namely EfficientNet-B1 and EfficientNet-B2, highlighted the delicate balance between model complexity and performance. While the latter variants exhibited slightly higher test accuracies, they incurred an increase in parameter count, presenting a trade-off that warrants consideration.

Ultimately, EfficientNet-B0 emerged as the optimal choice due to its favorable parameter-accuracy trade-off, reaffirming the rationale behind its selection as the pretrained model for this task. These findings underscore the significance of informed model selection and highlight the effectiveness of EfficientNet-B0 in addressing fine-grained classification challenges. Moving forward, the insights gained from this study can inform future research and application of EfficientNet models in various computer vision domains.

References

[1] Will Koehrsen. *Transfer Learning with Convolutional Neural Networks in PyTorch*. 2018. URL: https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce.

[2] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. URL: https://arxiv.org/abs/1905.11946.

[3] C. Wah et al. *Caltech-UCSD Birds-200-2011 (CUB-200-2011)*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011. URL: https://www.vision.caltech.edu/datasets/cub_200_2011/.